SPECIFICATION

METHOD AND MECHANISM FOR DIAGNOSING COMPUTER APPLICATIONS

USING TRACES

5 BACKGROUND AND SUMMARY

[0001] Tracing is an approach for logging the state of computer applications at different

points during its course of execution. Tracing is normally implemented by inserting

statements in the computer application code that outputs status/state messages ("traces") as

the statements are encountered during the execution of the code. Statements to generate

10 traces are purposely placed in the computer application code to generate traces

corresponding to activities of interest performed by specific sections of the code. The

generated trace messages can be collected and stored during the execution of the application

to form a trace log.

[0002] Programmers often use tracing and trace logs to diagnose problems or errors that

15 arise during the execution of a computer application. When such a problem or error is

encountered, trace logs are analyzed to correlate trace messages with the application code to

determine the sequence, origin, and effects of different events in the systems and how they

impact each other. This process allows analysis/diagnoses of unexpected behavior or

programming errors that cause problems in the application code.

20 [0003] In a parallel or distributed environment, there are potentially a number of distributed

network nodes, with each node running a number of distinct execution entities such as

threads, tasks or processes (hereinafter referred to as "threads"). In many modern computer

1

EL846343437US
Alok Srivastava et al.

applications, these threads perform complex interactions with each other, even across the network to threads on other nodes. Often, each of the distributed nodes maintains a separate log file to store traces for their respective threads. Each distributed node may also maintain multiple trace logs corresponding to separate threads on that node.

5 [0004] Diagnosing problems using multiple trace logs often involves a manual process of repeatedly inspecting different sets of the trace logs in various orders to map the sequence and execution of events in the application code. This manual process attempts to correlate events in the system(s) with the application code to construct likely execution scenarios that identify root causes of actual or potential execution problems. Even in a modestly

10 distributed system of a few nodes, this manual process comprises a significantly complex task, very much limited by the capacity of a human mind to comprehend and concurrently analyze many event scenarios across multiple threads on multiple nodes. Therefore, analyzing traces to diagnose applications in parallel and/or distributed systems is often a time consuming and difficult exercise fraught with the potential for human limitations to

15 render the diagnoses process unsuccessful. In many cases, the complexity of manual trace analysis causes the programmer to overlook or misdiagnose the real significance of events captured in the trace logs. With the increasing proliferation of more powerful computer systems capable of greater execution loads across more nodes, the scope of this problem can only increase.

20 [0005] The present invention is directed to a method and mechanism for improved diagnoses of computer systems and applications using tracing. According to an aspect of one embodiment of the invention, trace messages are materialized using a markup language

syntax. Hyperlinks can be placed in the trace messages to facilitate navigation between sets of related traces. Specific traces or portions of traces can be emphasized using markup language tools to highlight text. Another aspect of an embodiment of the invention pertains to a method and mechanism for generating trace messages in a markup language syntax.

5   Further aspects, objects, and advantages of the invention are described below in the detailed description, drawings, and claims.

3

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The accompanying drawings are included to provide a further understanding of the invention and, together with the Detailed Description, serve to explain the principles of the invention.

5 [0007] Fig. 1 shows an example of a communications operation between two network nodes and corresponding trace logs.

[0008] Fig. 2 shows trace logs including traces in markup language pseudocode according to an embodiment of the invention.

[0009] Fig. 3 is a diagram of a system for using and generating traces having markup

10 language syntax according to an embodiment of the invention.

[0010] Fig. 4 is a flowchart of a process for materializing traces with markup language syntax according to an embodiment of the invention.

[0011] Figs. 5 and 6 are diagrams of system architectures with which the present invention may be implemented.

4

DETAILED DESCRIPTION

[0012] The present invention is disclosed in an embodiment as a method and mechanism for implementing tracing and trace logs. The disclosed embodiment of the invention is directed to trace logs for distributed and parallel systems. However, the principles presented here are

5 equally applicable to trace log(s) in other system architecture configurations, including single node configurations, and thus the scope of the invention is not to be limited to the exact embodiment shown herein.

[0013] An aspect of one embodiment of the present invention is directed to traces comprising markup language syntax. A markup language is a collected set of syntax definitions that describes the structure and format of a document page. A widely used

10 markup language is the Standard Generalized Markup language ("SGML"). A common implementation of SGML is the HyperText Markup Language ("HTML"), which is a specific variant of SGML used for the world wide web. The Extensible Markup Language ("XML") is another variant of SGML. For explanatory purposes only, the invention is

15 described using HTML-compliant markup language syntax. However, it is noted that the present invention is not limited to any specific markup language syntax, but is configurable to work with many markup languages.

[0014] Analysis of traces is greatly facilitated, pursuant to an embodiment, by using traces implemented with markup language syntax. To illustrate this aspect of the invention,

20 consider a simple communications operation that is performed between two network nodes. Fig. 1 shows Nodes 1 and 2 executing an operation that consists of a first message that is sent from Node 1 to Node 2 and a response message that is subsequently sent from Node 2

to Node 1. Assume that a first trace log 100 maintains traces for threads executing on Node 1 and a second trace log 102 maintains traces for threads executing on Node 2.

[0015] When analyzing trace logs for communications operations that send messages between network nodes, it is common for sets of related traces to appear in multiple trace logs across the network. For example, "send" operation trace in a first trace log at a first node often has a counterpart "receive" operation trace located in a second trace log at a second node. Thus in the example of Fig. 1, the trace for the initial "send" operation from Node 1 to Node 2 is stored in log file 100 as trace message 5000. The trace for the "receive" operation that occurs on Node 2 is stored in trace log 102 as trace message 10000. The trace for the subsequent "send response" operation from Node 2 to Node 1 is stored as trace message 10200 in trace log 102. The trace for the "receive" operation that occurs at Node 1 for the response message is stored as trace message 22000 in trace log 100.

[0016] Consider if it is desired to analyze/diagnose this communications operation between Node 1 and Node 2. When a programmer analyzes the set of traces corresponding to that communications operation, it is likely that the programmer must review both the send and receive traces. In this example, the send and receive traces for the communications operation are spread across multiple trace logs on multiple nodes, and the traces of interest may be buried among hundreds or thousands of irrelevant traces that correspond to applications/operations of no immediate interest. Even in this very simple example, analysis of the trace logs could involve a complex and time-consuming task just to identify the traces of interest. That difficult task is compounded by the additional burden needed to manually jump between the different trace logs to chase the chain of traces across the multiple

6

network nodes. In the real world, this analysis/diagnosis task could become far more

difficult because of messaging operations that involve many more threads across many more

network nodes.

[0017] To address this problem, one embodiment of the present invention materializes trace

5   messages using a markup language syntax. By implementing trace messages using markup

language syntax, navigational intelligence can be embedded into the trace messages using

"hyperlinks." A hyperlink is an element in an electronic document or object that links to

another place in same document/object or to an entirely different document/object. As noted

above, when a programmer analyzes the set of traces corresponding to that communications

10   operation, it is likely that the programmer must review both the send and receive traces. For

this reason, it is useful to link related communications traces at the senders and receivers of

inter-nodal messages. Thus, a send trace is hyperlinked to its counterpart receive trace. The

hyperlinks can be defined in both the forward and reverse directions. A chain of linked

traces can be established whereby each trace is hyperlinked in sequential order to both its

15   predecessor and successor trace. All traces relating to a common operation or activity can

therefore be linked together via a chain of hyperlinks extending from a first trace through all

other related traces.

[0018] Fig. 2 depicts trace logs 200 and 202 that include traces messages implemented using

markup language pseudocode. Trace log 200 corresponds to trace log 100 of Fig. 1 and

20   contains traces generated by threads on Node 1. Trace log 202 corresponds to trace log 102

of Fig. 1 and contains traces for threads on Node 2. Each of the trace messages for the

communications operation shown in Fig. 1 are represented in Fig. 2 using markup language

pseudocode to illustrate hyperlinks between related trace messages. In particular, the "send" trace message 5000 in trace log 200 includes a forward hyperlink to its corresponding "receive" trace message 10000 in trace log 202. The "receive" trace message 10200 includes a reverse hyperlink to the "send" trace message 5000. The "send response" trace

5   message 10200 in trace log 202 is forward linked to its corresponding "receive response" trace message 22000 in trace log 200. The "receive response" trace message 22000 is reverse hyperlinked to the "send response" trace message 10200. While not a send and receive pair, trace messages 10000 and 10200 could be hyperlinked together to indicate the sequential nature of the operations corresponding to these traces. Any suitable markup

10   language syntax may be employed to implement this type of hyperlinking.

[0019] Once the trace messages have been materialized into a markup language syntax, any browser or viewer capable of interpreting the chosen markup language may be used to navigate the trace log(s). The traces for any activity of interest can be navigated by identifying one of the activity's traces and traversing the chain of hyperlinks extending from

15   that trace – without requiring any manual searching for related traces. Since both forward and reverse hyperlinks can be embedded into the trace log, the traces for an activity of interest can be traversed in both the forward or reverse directions.

[0020] Fig. 3 depicts a diagram of an embodiment of a system for utilizing and materializing trace logs that include markup language syntax. For the purposes of illustration, Fig. 3

20   shows a first node having a first trace log 304 and a second node 306 having a second trace log 308. According to this embodiment of the invention, trace messages generated by node 302 or 306 are initially parsed by a parser 311 to tokenize the information located in a trace

message string. That tokenized information is used to create intermediate data 313

corresponding to the trace messages, such as intermediate hyperlinking information that

relates a particular "send" trace with its corresponding "receive" trace. A mark-up language

converter mechanism 314 performs a conversion operation to materialize the traces from

5   trace logs 304 and 308 into converted trace logs 310 and 312 in a markup language syntax.

A suitable browser 316 can be employed to view the markup language information

embedded in trace logs 310 and 312.

[0021] According to an embodiment of the invention, trace messages from multiple trace

logs can be collected into a single trace log, rather than multiple materialized trace logs 310

10   and 312 as shown in Fig. 3. Thus, trace log 304 and trace log 308 can be combined into a

single set of trace messages. Many types of ordering can be employed to combine trace

messages from different trace files. For example, trace messages can be ordered in

time/date order. Trace messages can also be sorted based upon the specific resource or type

of operation corresponding to a trace message. In addition, specific subsets of trace

15   messages from trace logs 304 and 308 can be operatively selected for conversion into a

markup language syntax, to filter out traces that are of no interest.

[0022] The present invention also provides a method and mechanism for emphasizing

specific traces or portions of traces in a trace log using markup language syntax. According

to this aspect of the invention, traces or trace portions of particular interest include markup

20   language elements that provides visual emphasis when viewed in a suitable browser 316.

The visual emphasis may encompass any type of visual cue that differentiates one portion of

9

text from another portion of text, such as bolding certain areas of text, using different colors, using different fonts or font sizes, underlining, etc.

[0023] This aspect of the invention is useful if it is desired to emphasize traces or portions of traces corresponding to a unique characteristic. For example, consider if it is desired to

5   analyze or diagnose all operations performed against a specific system resource. This would involve identifying all traces that relate to that system resource. A search is performed against the trace logs to identify all traces corresponding to that system resource. During the conversion process, those identified traces would undergo a conversion to include additional markup language elements to visually separate those trace messages from other trace

10   messages. When viewed with browser 316, this "emphasized pattern" would readily highlight all traces corresponding to the system resource of interest. Moreover, hyperlinks can be embedded to permit sequential navigation through all the traces in the emphasized pattern of traces.

[0024] Thus, the invention includes a search or filter mechanism to search for particular

15   patterns in the trace logs. Conversion instructions are sent from a user interface to identify specific patterns that should be searched and will be either emphasized or filtered out. In an embodiment, the browser 316 includes an interface for a user to input a string or regular expression to be used for the filter/search procedure. The markup language converter 314 uses the results of the filter/search procedure to determine which trace messages require

20   markup language conversion and what types of conversions are necessary to emphasize patterns.

[0025] An embodiment of the invention for converting traces into a markup language format utilizes fixed format trace strings. The process for extracting information from a trace in this approach is driven by knowledge of the position and existence of specific data items in a trace string. For example, unique identifiers for events, operations or other types of data

5   objects are embedded at a recognized location(s) in the trace strings. Extracting these unique identifiers permits efficient correlation between related traces.

[0026] Fig. 4 depicts a flowchart of a process for converting traces from fixed format strings into a markup language format according to one embodiment of the invention. At 402, a trace message is received from a trace log. The trace message is parsed (404) to tokenize

10   and identify information in the trace string needed for the conversion process. Examples of information extracted from the trace string includes timestamps and unique identifiers. A determination is thereafter made regarding navigation patterns to be embedded into the converted trace messages (406). In effect, information extracted from each trace message is compared to information extracted from other trace messages to determine navigable

15   relationships between the traces. The determination is with respect to whether a trace message should be hyperlinked to any other trace messages.

[0027] For example, 1-to-1 communications involving send and receive pairs of traces are identified at this stage. Information is stored to identify these related traces as candidates for embedded hyperlinks when markup language conversions of the traces are generated. It is

20   noted that other types of communications relationships, including 1-to-many and many-to-many relationships, are also identified at this stage. As an example, a broadcast message is a message that is broadcast from a single node to possibly many nodes. This relationship is

11

also identified at step 406 and intermediate data is stored to distinguish these traces as candidates for hyperlinks when markup language version of the traces are materialized.

[0028] A filter or search condition may be established for the traces (408). A user desiring to view a particular emphasized pattern may establish such a filter/search condition. If a

5   filter out condition has been established (410), then a search is performed and any traces matching the filter condition are filtered from the group of traces to be converted into a markup language format (412). Information extracted from the trace string during the parse procedure is used to determine if the trace string should be filtered. For example, a filter may be set to exclude all traces corresponding to a system resource "A". If a trace message

10   corresponding to this filter condition is encountered, then the trace message will be discarded from the conversion process and will not be viewed by the browser 316. If the trace message does not correspond to the filter condition, then the conversion process proceeds for that trace message.

[0029] If a search condition has been established for a desired emphasized pattern (414),

15   then a search is performed and any traces matching the search condition are identified as candidates for additional markup language elements to include differentiating visual cues for conversion into a markup language format (416). Information extracted from the trace string during the parse procedure is again used to determine if the trace string should be emphasized.

20   [0030] The traces are thereafter materialized in a markup language format (418). In particular, traces that are part of navigable patterns are materialized to include hyperlinks. Traces that are part of emphasized patterns are materialized to include markup language

12

elements to provide additional visual cues. The materialized traces in markup language

format can be viewed using any suitable browser compatible with the particular markup

language used for the conversion.

[0031] The following represents an example of a generic template that can be used for a

5   fixed trace string:

Generic Template: <Header> <keyword> <arg0> <arg1> <arg2> <arg3> ... <argn>

In this generic template, <Header> represents the portion of the trace string containing

10   required data items used for the conversion process. <Keyword> represents one or more

keyword "hints" that provide additional information regarding the format/type of arguments

that follow. <arg0> through <argn> represents additional arguments to be generated with

the trace string.

[0032] As a more specific example of a fixed format trace string, consider the following

15   trace string which is generated for a database operation during a deadlock detection ("DD")

search by a distributed lock manager ("DLM"):

7C839FEF:00000010   5   4 10435 51 DLM-DD start:ddTS[0.1] [TXN]res
[0x1][0x1],[TX], node 0

20

In this example trace string, the <header> portion includes the following information:

7C839FEF:00000010   5   4 10435 51

This is a fixed format header record representing the following items of information:

<timestamp> <sid> <pid> <event> <opcode>

where,

5

a) <timestamp> is "7C839FEF:00000010", which represents the timestamp for the trace;

b) <sid> is "5", which represents a "session id" for the particular computer/database session in which the operation corresponding to the trace is executed;

c) <pid> is "4" and which represents a particular process id;

10

d) <event> is "10435", which represents an event identifier that allows sets of related operation for an event to be correlated;

e) <opcode> is "51", which represents an operation identifier for subgrouping within an event that is related to a particular operation.

15

[0033] In the example trace string, "DLM-DD" represents a <keyword> that provides a "hint" regarding the type of operation performed and the type/format of the arguments that follow.

[0034] <arg0> is represented by the string "start:". The "start:" value identifies a particular operation or stage of an operation that is performed. Other examples of types of information

20

that may be included in <arg0> for a deadlock detection operation are: "send:--", "receive:--", "found:--", "confirm:--", "drop (victim done):--"

[0035] <arg1> is represented by the "ddTS[x.y]" string where x and y are integers which maintain the deadlock search count. The "ddTS[0.1]" value in the example string is a token number that allows related traces (e.g., send-receive pairs) to be identified across multiple nodes.

5 [0036] <arg2> is represented by the "[TXN]res | [PROC]res" string which specified whether it is a process (PROC) owned resource or a transaction (TXN) owned resource.

[0037] <arg3> has the value "node 0" to indicate a particuylar node related to the trace message.

10 [0038] The following represents a series of example trace messages generated for a deadlock detection operation by a distributed lock manager:

Traces from Node 0

    7C839FEF:00000010   5   4 10435 51 DLM-DD start:ddTS[0.1] [TXN]res

15 [0x1][0x1],[TX], node 0

    7C839FEF:00000011   5   4 10435 52 DLM-DD send:ddTS[0.1] [TXN]res, dest node 3

20 Traces from Node 3

    7C839FEF:00000050   5   4 10435 52 DLM-DD receive:ddTS[0.1] [TXN]res, src node 0

15

7C839FEF:00000051   5   4 10435 53 DLM-DD found:ddTS[0.1] [TXN]res, valid start, node 3

5 7C839FEF:00000052   5   4 10435 54 DLM-DD confirm:ddTS[0.1] [TXN]res, origin victim, node 3

7C839FEF:00000014   5   4 10435 55 DLM-DD drop (victim done):ddTS[0.1] [TXN]res, node 3

10 [0039] In the example traces above, the send operation (trace 2 from node 0) and receive operation (trace 1 from node 3) pair form a direct linking pattern where opcode "52" is used to create the link identification. Thus, conversion into a markup language format would result in a hyperlink between these two traces. The following is an example of the conversion of the send operation trace in a HTML-based markup language format:

15 <HTML>

<BODY>

7c839FEF:00000010 5 4 10435 51 DLM-DD start:ddTS [0,1] [TXN] res [0x1] [0x1], [TX], n...ode0

20 <A HREF= "trace_node3.html#52" NAME= "52">

7C839FEF:00000011 5 4 10435 52 DLM-DD send:ddTS [0,1] [TXN] res, dest node 3

</A>

</BODY>

</HTML>

[0040] The following is another example of converting the receive operation trace into a

HTML-based markup language format:

<HTML>

5   <BODY>

<A HREF= "trace_node0.html#52" NAME= "52">

7C89FEF:00000050 5 4 10435 52 DLM-DD receive:ddTS[0,1] [TXN]res, src node 0

</A>

10   7C839FEF:00000051 5 4 10435 53 DLM-DD found:ddTS[0,1] [TXN]res, valid

start, node...ode 3

7C839FEF:00000014 5 4 10435 55 DLM-DD drop (victim done) :ddTS[0,1]

[TXN]res, node...3

15   </BODY>

</HTML>

[0041] In the examples above, traces for a particular deadlock detection operation can also

be found with keyword DLM-DD as the primary key and arg1 (ddTS[x.y]) as the seconadry

key.

20   [0042] In the above example, a possible emphasizing pattern that can be identified could be

to establish a search filter for all traces for transaction based resources grouped by string

[TXN]res. If it is desired to emphasize this pattern, the following is an example of a

converted markup language format for these traces:

<HTML>

<BODY>

5   <B>7C839FEF:00000010 5 4 10435 51 DLM-DD start:ddTS[0,1] [TXN] res [0x1]

[0x1], [TX...], node0</B>

<

<B>7C839FEF:00000011 5 4 10435 52 DLM-DD send:ddTS[0,1] [TXN]res, dest

node 3</B>

10   </BODY>

</HTML>

[0043] The following is another example of a converted trace having this emphasized

pattern:

<HTML>

15   <BODY>

<B>

7C89FEF:00000050 5 4 10435 52 DLM-DD receive:ddTS[0,1] [TXN]res, src node 0

</B>

20   <B>

7C839FEF:00000051 5 4 10435 53 DLM-DD found:ddTS[0,1] [TXN]res, valid

start, node...3

18

</B>

<B>

7C839FEF:00000052 5 4 10435 54 DLM-DD confirm:ddTS[0,1] [TXN]res, origin

5   victim, n...ode 3

</B>

<B>

7C839FEF:00000 5 4 10435 55 DLM-DD drop (victim done) :ddTS[0,1] [TXN] res,

10   node...3

</B>

</BODY>

</HTML>

15   **SYSTEM ARCHITECTURE OVERVIEW**

[0044] Referring to Fig. 5, in an embodiment, a computer system 520 includes a host

computer 522 connected to a plurality of individual user stations 524. In an embodiment,

the user stations 524 each comprise suitable data terminals, for example, but not limited to,

e.g., personal computers, portable laptop computers, or personal data assistants ("PDAs"),

20   which can store and independently run one or more applications, i.e., programs. For

purposes of illustration, some of the user stations 524 are connected to the host computer

522 via a local area network ("LAN") 526. Other user stations 524 are remotely connected

19

to the host computer 522 via a public telephone switched network ("PSTN") 528 and/or a

wireless network 530.

[0045] In an embodiment, the host computer 522 operates in conjunction with a data storage

system 531, wherein the data storage system 531 contains a database 532 that is readily

5   accessible by the host computer 522. Note that a multiple tier architecture can be employed

to connect user stations 524 to a database 532, utilizing for example, a middle application

tier (not shown). In alternative embodiments, the database 532 may be resident on the host

computer, stored, e.g., in the host computer's ROM, PROM, EPROM, or any other memory

chip, and/or its hard disk. In yet alternative embodiments, the database 532 may be read by

10   the host computer 522 from one or more floppy disks, flexible disks, magnetic tapes, any

other magnetic medium, CD-ROMs, any other optical medium, punchcards, papertape, or

any other physical medium with patterns of holes, or any other medium from which a

computer can read. In an alternative embodiment, the host computer 522 can access two or

more databases 532, stored in a variety of mediums, as previously discussed.

15   [0046] Referring to Fig. 6, in an embodiment, each user station 524 and the host computer

522, each referred to generally as a processing unit, embodies a general architecture 605. A

processing unit includes a bus 606 or other communication mechanism for communicating

instructions, messages and data, collectively, information, and one or more processors 607

coupled with the bus 606 for processing information. A processing unit also includes a main

20   memory 608, such as a random access memory (RAM) or other dynamic storage device,

coupled to the bus 606 for storing dynamic data and instructions to be executed by the

processor(s) 607. The main memory 608 also may be used for storing temporary data, i.e.,

20

variables, or other intermediate information during execution of instructions by the

processor(s) 607. A processing unit may further include a read only memory (ROM) 609 or

other static storage device coupled to the bus 606 for storing static data and instructions for

the processor(s) 607. A storage device 610, such as a magnetic disk or optical disk, may

5   also be provided and coupled to the bus 606 for storing data and instructions for the

processor(s) 607.

[0047] A processing unit may be coupled via the bus 606 to a display device 611, such as,

but not limited to, a cathode ray tube (CRT), for displaying information to a user. An input

device 612, including alphanumeric and other columns, is coupled to the bus 606 for

10   communicating information and command selections to the processor(s) 607. Another type

of user input device may include a cursor control 613, such as, but not limited to, a mouse, a

trackball, a fingerpad, or cursor direction columns, for communicating direction information

and command selections to the processor(s) 607 and for controlling cursor movement on the

display 611.

15   [0048] According to one embodiment of the invention, the individual processing units

perform specific operations by their respective processor(s) 607 executing one or more

sequences of one or more instructions contained in the main memory 608. Such instructions

may be read into the main memory 608 from another computer-usable medium, such as the

ROM 609 or the storage device 610. Execution of the sequences of instructions contained in

20   the main memory 608 causes the processor(s) 607 to perform the processes described herein.

In alternative embodiments, hard-wired circuitry may be used in place of or in combination

21

with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and/or software.

[0049] The term "computer-usable medium," as used herein, refers to any medium that provides information or is usable by the processor(s) 607. Such a medium may take many

5   forms, including, but not limited to, non-volatile, volatile and transmission media. Non-volatile media, i.e., media that can retain information in the absence of power, includes the ROM 609. Volatile media, i.e., media that can not retain information in the absence of power, includes the main memory 608. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise the bus 606. Transmission media

10   can also take the form of carrier waves; i.e., electromagnetic waves that can be modulated, as in frequency, amplitude or phase, to transmit information signals. Additionally, transmission media can take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[0050] Common forms of computer-usable media include, for example: a floppy disk,

15   flexible disk, hard disk, magnetic tape, any other magnetic medium, CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, RAM, ROM, PROM (i.e., programmable read only memory), EPROM (i.e., erasable programmable read only memory), including FLASH-EPROM, any other memory chip or cartridge, carrier waves, or any other medium from which a processor 607 can retrieve

20   information. Various forms of computer-usable media may be involved in providing one or more sequences of one or more instructions to the processor(s) 607 for execution. The

22

instructions received by the main memory 608 may optionally be stored on the storage

device 610, either before or after their execution by the processor(s) 607.

[0051] Each processing unit may also include a communication interface 614 coupled to the

bus 606. The communication interface 614 provides two-way communication between the

5   respective user stations 524 and the host computer 522. The communication interface 614

of a respective processing unit transmits and receives electrical, electromagnetic or optical

signals that include data streams representing various types of information, including

instructions, messages and data. A communication link 615 links a respective user station

524 and a host computer 522. The communication link 615 may be a LAN 526, in which

10   case the communication interface 614 may be a LAN card. Alternatively, the

communication link 615 may be a PSTN 528, in which case the communication interface

614 may be an integrated services digital network (ISDN) card or a modem. Also, as a

further alternative, the communication link 615 may be a wireless network 530. A

processing unit may transmit and receive messages, data, and instructions, including

15   program, i.e., application, code, through its respective communication link 615 and

communication interface 614. Received program code may be executed by the respective

processor(s) 607 as it is received, and/or stored in the storage device 610, or other associated

non-volatile media, for later execution. In this manner, a processing unit may receive

messages, data and/or program code in the form of a carrier wave.

20   [0052] In the foregoing specification, the invention has been described with reference to

specific embodiments thereof. It will, however, be evident that various modifications and

changes may be made thereto without departing from the broader spirit and scope of the

23

invention. For example, the reader is to understand that the specific ordering and

combination of process actions shown in the process flow diagrams described herein is

merely illustrative, and the invention can be performed using different or additional process

actions, or a different combination or ordering of process actions. The specification and

5   drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.

24